Powering Up: Lab Automation with Labgrid and CI

Trevor Gamblin, Tim Orling

Agenda

- Who We Are
- Background Context
- Walkthrough: The Single-Developer Labgrid Loop
- Time to Automate: Continuous Integration with Hardware-in-the-Loop
- Scaling Up
- Challenges
- Future Work
- Questions?
- (A Very Extensive) Appendix

Reference links for this talk: here

Who We Are

Trevor:

- Embedded Linux Developer at BayLibre
- Yocto Project contributor, maintainer (Python recipes, Patchtest)
- **Main Interests:** Automation, Python, Kernel dev, RISC-V, Rust

Tim:

- Principal Software Engineer at Konsulko Group
- OpenEmbedded Board member, OpenEmbedded TSC member, Yocto Project contributor, maintainer (Python recipes and more)
- **Main Interests:** Automation, Security and encryption, Python

Background Context

Impetus: Kevin's #1 Use-Case

Automating u-boot netboot:

1. power cycle
2. break into u-boot
3. TFTP kernel, dtb, ramdisk
4. setup kernel command-line
5. boot

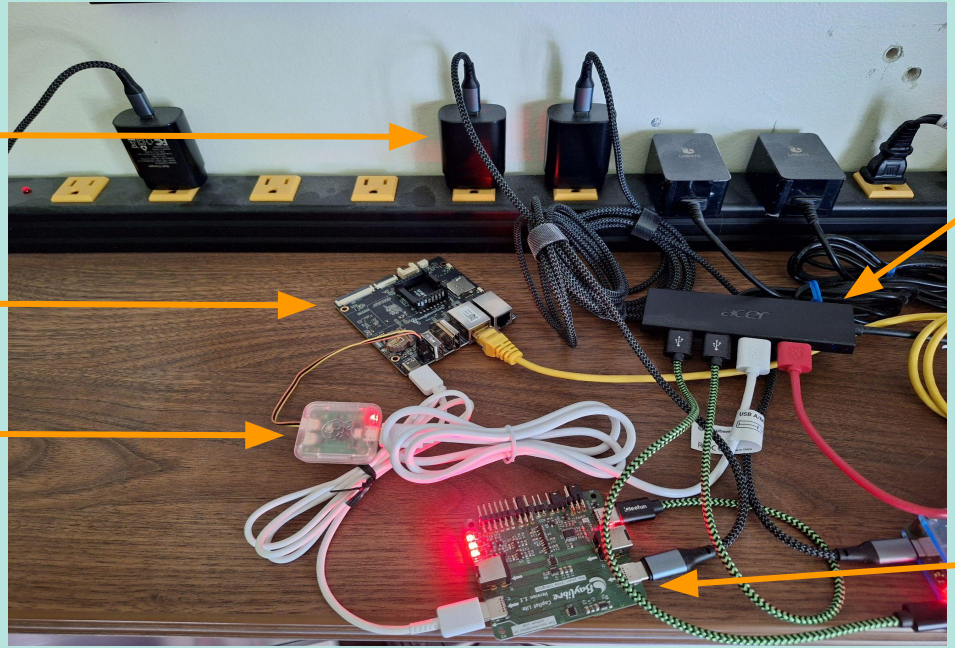
Kevin's questions:

1. Where do I configure u-boot params (e.g. load addrs) and kernel command-line?
2. How does this scale from 1 board to 50?

GOAL: Automate: Interfacing With/Controlling Hardware

- **Automating your workflow -> fewer mistakes during repetitive tasks**
- We don't want to be concerned with manual access steps when we're writing code!
- How can we approximate the same workflow for many different development situations? **Use labgrid!**
- How do we minimize differences with CI setups? **Use labgrid!**
- We struggled with it at first - we're not the experts - but maybe what we've learned will be useful to you
- More than we can cover in 30 minutes - some in the Appendix

Hardware: Trevor's Setup



The details of this setup could be abstracted away…

Preview: Two Setups, One Command

Our hardware setups (see Appendix for full details):

Component	Trevor	Tim
USB-to-Serial Cable	RPi Debug Probe	Silicon Labs CP210x UART Bridge
Power Control	BayLibre Copilot (i.e. GPIO Switch, USB-C Charger Power Passthrough)	Netgear GS305EPP PoE Managed Switch
SD Card Mux	None	SD Wire (for distro flashing)
Development Board	BeaglePlay	BeaglePlay

Both tftpboot to console with: `labgrid-client --config ecogrid-client.yaml -s shell con`

Walkthrough

The Single-Developer Labgrid Loop (SDLL)

Defining a Resource Configuration (Part 1)

- Host system setup: Appendix
- Define resources (devices) used to interact with the target in **ecogrid-exporter.yaml**
- Need to know how the system sees them (**labgrid-suggest** is helpful for pluggables)
- Example: We have a device that can be matched as a USBSerialPort, with:
  - **ID_SERIAL_SHORT:** E663AC91D35C5B2E
  - **ID_USB_INTERFACE_NUM:** '01' (useful if the device has more than one port)
- With **bp-serial-port** as a resource label, this might look like:

Group label for resources

Resource location (user reference, not used by labgrid. See here)

Specific **resource label**

```
beagleplay:
  location: ecogrid
  bp-serial-port:
    cls: 'USBSerialPort'
    match:
      '@ID_SERIAL_SHORT': 'E663AC91D35C5B2E'
```

Resource **class indicator**

Device match pattern (multiple possible options, from udev)

Defining a Resource Configuration (Part 2)

Trevor's final ecogrid-exporter.yaml

```
beagleplay:
  location: ecogrid
  bp-power-gpio:
      cls: 'MatchedSysfsGPIO'
      match:
      '@SUBSYSTEM': usb
      '@ID_SERIAL_SHORT': 'D30HF051'
      pin: 0
  bp-serial-port:
      cls: 'USBSerialPort'
      match:
      '@ID_SERIAL_SHORT': 'E663AC91D35C5B2E'
  bp-tftp:
      cls: 'RemoteTFTPProvider'
      host: '192.168.40.134'
      internal: '/srv/tftp/beagleplay/'
      external: 'beagleplay/'
```

Tim's final beagleplay-exporter.yaml config:

```
beagleplay-1:
  location: 'c4labs-chassis-1'
  bp-serial-port:
    cls: 'USBSerialPort'
    match:
      ID_PATH:
'platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.4:1.0'
  bp-sdwire:
    cls: 'USBSDWireDevice'
    match:
      ID_SERIAL_SHORT: '202203010023'
  bp-power:
    cls: 'PDUDaemonPort'
    host: 'beagleplay-raspberrypi4-64.local'
    pdu: 'netgearplus'
    index: 3
  …
```

Using the Resources - Creating a Place

- **Run: labgrid-coordinator** - process for managing resources, handle access, etc.
- **Run: labgrid-exporter ecogrid-exporter.yaml** - make resources available to labgrid
- Create a place and match against it by running:
  a. **labgrid-client --place beagleplay create**
  b. **labgrid-client -p beagleplay add-match */beagleplay/***
- **Run: labgrid-client -p beagleplay acquire** - select and lock the new place
- Now we see:

```
(labgrid-venv) tgamblin@ecogrid:~/labgrid$ labgrid-client resources
ecogrid/beagleplay/NetworkSysfsGPIO
ecogrid/beagleplay/NetworkSerialPort
ecogrid/beagleplay/RemoteTFTPProvider
```

Exporter hostname, **not** “location” value in ecogrid-exporter.yaml

Using the Resources - Manual Interaction

- What if we want to connect to the BeaglePlay via serial port, and control the power?
- Answer: **labgrid-client**

```
(labgrid-venv) tgamblin@ecogrid:~/labgrid$ labgrid-client -p beagleplay console
connecting to NetworkSerialPort(target=Target(name='beagleplay', env=None), name='bp-serial-port',
state=<BindingState.bound: 1>, avail=True, host='ecogrid', port=38279, speed=115200, protocol='rfc2217')
calling telnet ecogrid 38279
Trying 127.0.1.1...
Connected to ecogrid.
Escape character is '^]'.
```

```
(labgrid-venv) tgamblin@ecogrid:~/labgrid$ labgrid-client -p beagleplay io high
(labgrid-venv) tgamblin@ecogrid:~/labgrid$
```

```
…

Debian GNU/Linux 12 BeagleBone ttyS2

BeagleBoard.org Debian Bookworm Minimal Image 2025-03-05
Support: https://bbb.io/debian
default username is [beagle]

BeagleBone login:
```

Using the Resources - Drivers and Strategies (Part 1)

- **Reminder: Goal: Automate Interfacing With/Controlling Hardware**
- Manual is nice, but it's not that much better than doing it without labgrid… **the real power-up is still out there**
  - Main benefit at this stage: **abstraction** - e.g. no need to reference /sys/class/gpio directly
- Need to define an environment file (e.g. **ecogrid-client.yaml**) for **labgrid-client** to know **how each resource is to be used**
- Each driver has valid kinds of resources it can bind to (listed in docs), e.g. a SerialDriver can bind to a NetworkSerialPort, RawSerialPort, or **USBSerialPort (our case)**
  - Many also have settings that the user can customize

Using the Resources - Drivers and Strategies (Part 2)

- Here's an example of how we write **ecogrid-client.yaml** (full version: here):

```
targets:
  main:
      resources:
        RemotePlace:
          name: beagleplay
      drivers:
        - SerialDriver:
            name: 'bp-console'
            bindings:
              port: 'bp-serial-port'
        - GpioDigitalOutputDriver:
            name: 'bp-power-gpio'
            bindings:
               gpio: 'bp-power-gpio'
        - DigitalOutputPowerDriver:
            name: 'bp-power'
            bindings:
              output: 'bp-power-gpio'
...
```

Target label ("main" works if it's the only one)

Specify the place we're interacting with

Array of driver definitions

Binding tells the driver which object to bind to (only essential if you have multiple resources/drivers of same class)

Some drivers can bind to other drivers - such as here, where we allow ourselves to toggle a GPIO like a power switch

NOTE: Some drivers e.g. UBootDriver can take multiple parameters, which we need to customize our boot process

Using the Resources - Drivers and Strategies (Part 3)

- The real power is in the **UBootDriver:**

```
...
      - UBootDriver:
            name: 'bp-uboot'
            bindings:
              console: 'bp-console'
            prompt: '=> '
            autoboot: 'Press SPACE to abort autoboot in 2 seconds'
            interrupt: ' '
            boot_commands:
              my_tftp: 'setenv bootargs "$bootargs console=ttyS2,115200n8
root=/dev/mmcblk0p3 ro rootfstype=ext4 resume=/dev/mmcblk0p2 rootwait
net.ifnames=0"; setenv kernel_comp_addr_r 0x85000000; setenv kernel_comp_size
0x2000000; tftp $kernel_addr_r beagleplay/Image.gz; tftp $fdt_addr_r
beagleplay/k3-am625-beagleplay.dtb; booti $kernel_addr_r - $fdt_addr_r'
```

Binds to bp-console (SerialDriver in previous slide)

Prompt matching options

List of possible boot commands to select from strategy

Essentially: get dtb and kernel image from ‘beagleplay/’ subfolder in tftp server, then boot the board with those + the eMMC rootfs

Using the Resources - Drivers and Strategies (Part 4)

- **Reminder: Goal: Automate Interfacing With/Controlling Hardware**
  - **In practical terms:** reliably transition the BeaglePlay from poweroff to shell via tftpboot, **with our custom kernel image** - need a **strategy** to glue it together
    - You'll usually want to write this yourself (Python)
- Example is here - the important things:
  - A **Status** class (really, enum) of possible states to match against
  - A **Strategy** class (e.g. **BeagleplayBootStrategy**) - contains binding info, and state machine (**transition()** method) for checking the status and making appropriate changes
  - For switching to shell:

```
elif status == Status.shell:
    # transition to uboot
    self.transition(Status.uboot)
    self._set_server_ip()
    self.uboot.boot("my_tftp")
    self.uboot.await_boot()
    self.target.activate(self.shell)
```

Helper function - sets up tftp-related networking in uboot shell

Transition to 'uboot' status first, so we can boot with our custom kernel

Run "tftp" command from boot_commands list in UBootDriver definition

Using the Resources - Taking Control!

- Via the Python interpreter (i.e. scripting proof-of-concept):

```
from labgrid import Environment


e = Environment("ecogrid-client.yaml")
t = e.get_target("main")
s = t.get_strategy("BeagleplayBootStrategy")
s.transition("shell")
```

- Or even simpler, using **labgrid-client**:

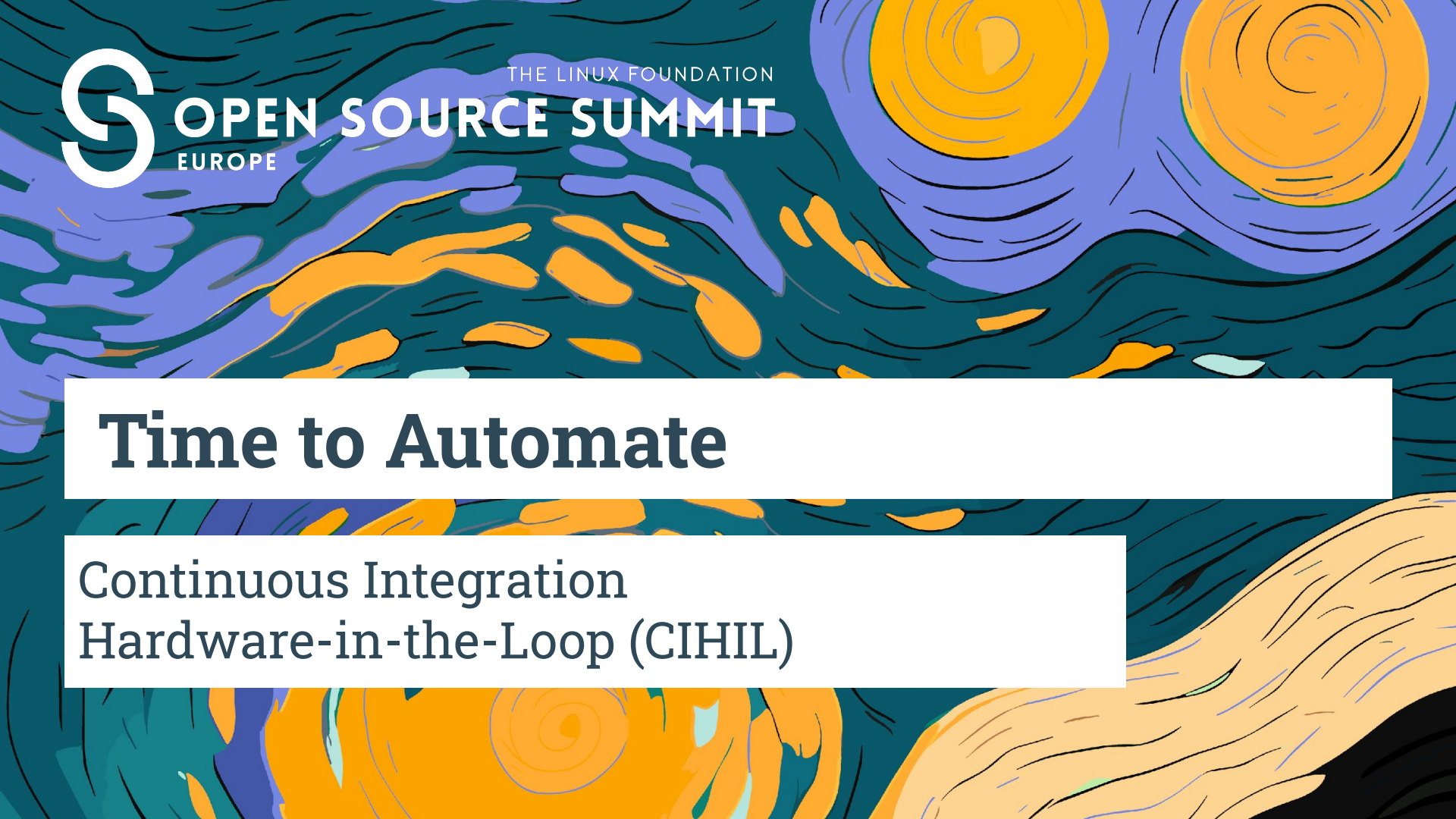
```
labgrid-client --config ecogrid-client.yaml -s shell con
```

Start console session upon completion

Tell labgrid which driver setup to use (including imported strategy)

Transition to a state (in this case, 'shell')

This is a huge, reusable improvement for our lab… but what about CI?



Time to Automate

Continuous Integration Hardware-in-the-Loop (CIHIL)

Time to Automate (Part 1): Pytest

```
testing
├── labgrid
│   ├── beagleplay-isar-labgrid-env.yaml
│   └── strategy-beagleplay-isar.py
├── pytest
│   ├── conftest.py
│   ├── test_00_userspace.py
│   ├── test_01_uname.py
│   └── test_02_kselftest_dt.py
└── pytest.ini
```

```
import json
import re

from labgrid.driver import ExecutionError

def test_tools_available(shell):
    tools = ['/bin/bash', '/usr/bin/ls']
    missing = []

    for tool in tools:
        stdout, stderr, code = shell.run(f"test -x {tool}")

        if code != 0:
            missing.append(tool)

    assert(missing == [])
```

```
[pytest]
log_format = %(asctime)s %(levelname)s %(message)s
log_date_format = %Y-%m-%d %H:%M:%Stry:
testpaths =
    pytest
```

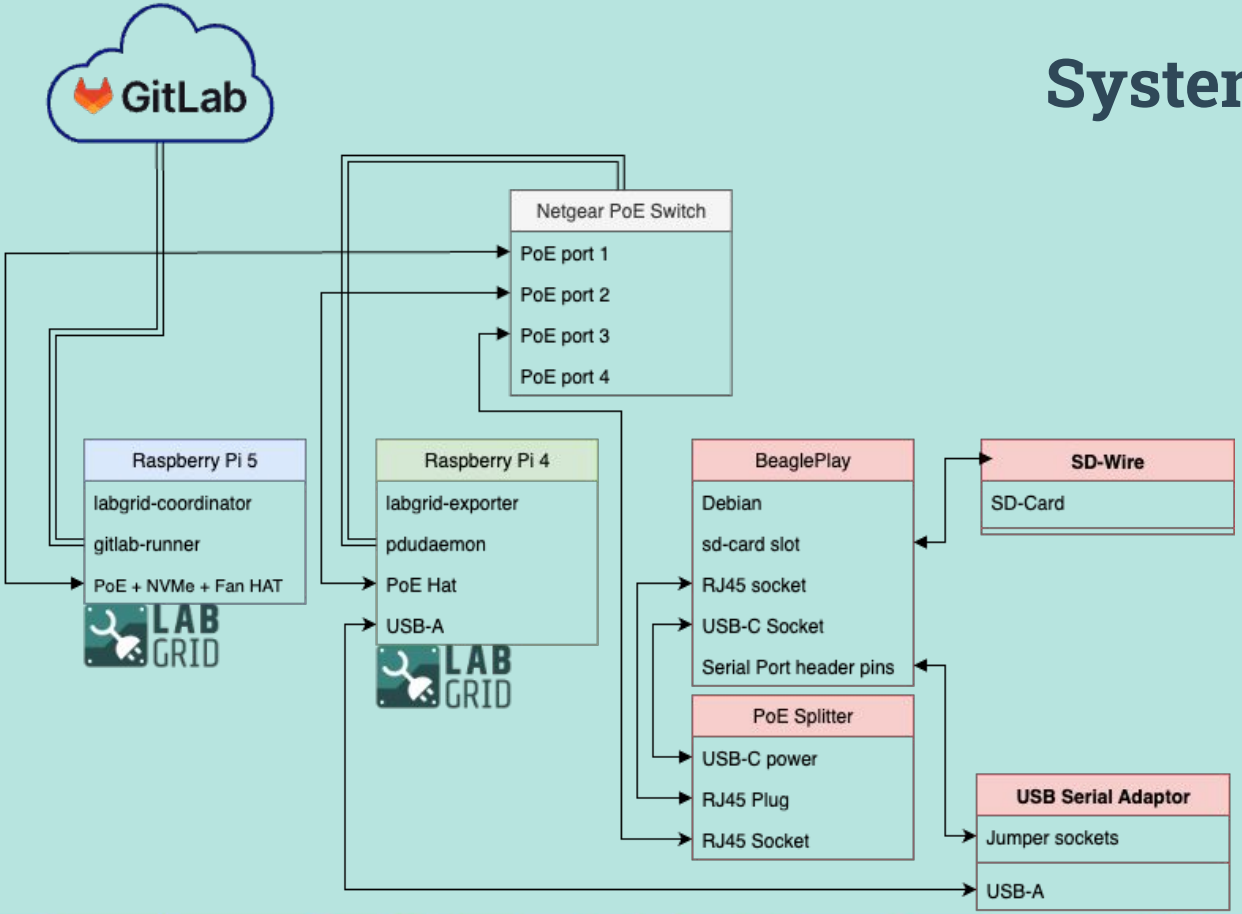
Repo: here

Time to Automate (Part 2): CI With GitLab

Process:

- GitLab CI Setup
- Build the kernel and device-tree (and perhaps ramdisk)
- Push those artifacts to tftp-server
- Launch labgrid job to boot with the new kernel and dtb via u-boot/tftp
- Run some simple tests to login and make sure the kernel is ok
- Run kselftest DT subset

System Diagram



Time to Automate (Part 3)

```
build-kernel:
  <<: *build_base
  stage: build
  script:
    …
    - cp beagleplay_defconfig linux-$KERNEL_VERSION/arch/arm64/configs/
    - cd linux-$KERNEL_VERSION
    - export CROSS_COMPILE=aarch64-linux-gnu-
    - make ARCH=arm64 beagleplay_defconfig && make ARCH=arm64 -j$(nproc) Image.gz
dtbs modules 2>&1 | tee $LOG
  parallel:
    matrix:
      - RELEASE: ['stable', 'mainline']
  artifacts:
    paths:
    - ${DESTDIR}/${KERNEL_IMAGE}
    - ${DESTDIR}/${DTB_FILE}
```

https://gitlab.com/moto-timo/build-beagleplay-kernel/-/blob/main/.gitlab-ci.yml?ref_type=heads

Time to Automate (Part 4)

```
# Hardware testing with labgrid
test-beagleplay-hardware:
  stage: test-hardware
  dependencies:
    - build-kernel
  before_script:
    - echo "Copying artifacts to tftp server"
    - cp public/$RELEASE/$KERNEL_IMAGE /srv/tftp/beagleplay/$RELEASE/$KERNEL_IMAGE
    - cp public/$RELEASE/$DTB_FILE /srv/tftp/beagleplay/$RELEASE/$DTB_FILE
    - echo "Setting up labgrid virtualenv"
    - virtualenv -p python3 testing/labgrid-venv
    - source testing/labgrid-venv/bin/activate
    - pip install --upgrade pip
    - pip install labgrid pytest_robotframework
  script:
    - echo "Starting BeaglePlay hardware tests with labgrid"
    - cd testing
    - ./run
```

https://gitlab.com/moto-timo/build-beagleplay-kernel/-/blob/main/.gitlab-ci.yml?ref_type=heads

Why Do We Need All of This?

- **<u>Reminder:</u> Goal: Automate Interfacing With/Controlling Hardware**
- Do you really want to do all of this manually? (NO!)
- **Expressive way to build out test infrastructure**
- KernelCI/Yocto - extra level of test automation becomes available
  - e.g. Trevor's nightly poky genericarm64 core-image-full-cmdline build
  - e.g. Tim's isar build and Yocto distro genericarm64 build
- One question remains (Kevin's #2): **How do we scale up?**
  - Answer: **It depends!**

Scaling Up

Scaling Up, Part 1: Config Changes

- This is where things start to depend more on your lab
- The docs cover writing config files well
- New targets mean:
  - Likely changing ‘main’ in client config to something else (if you have multiple targets in one test scenario)
  - Renaming resource aliases, driver bindings, etc. to avoid collisions
- Can use **jinja2** to generate config for many resources (see labgrid docs here)
  - Especially useful if most places will have very similar resources

What if we want to make multiple different test flows available to the same target?

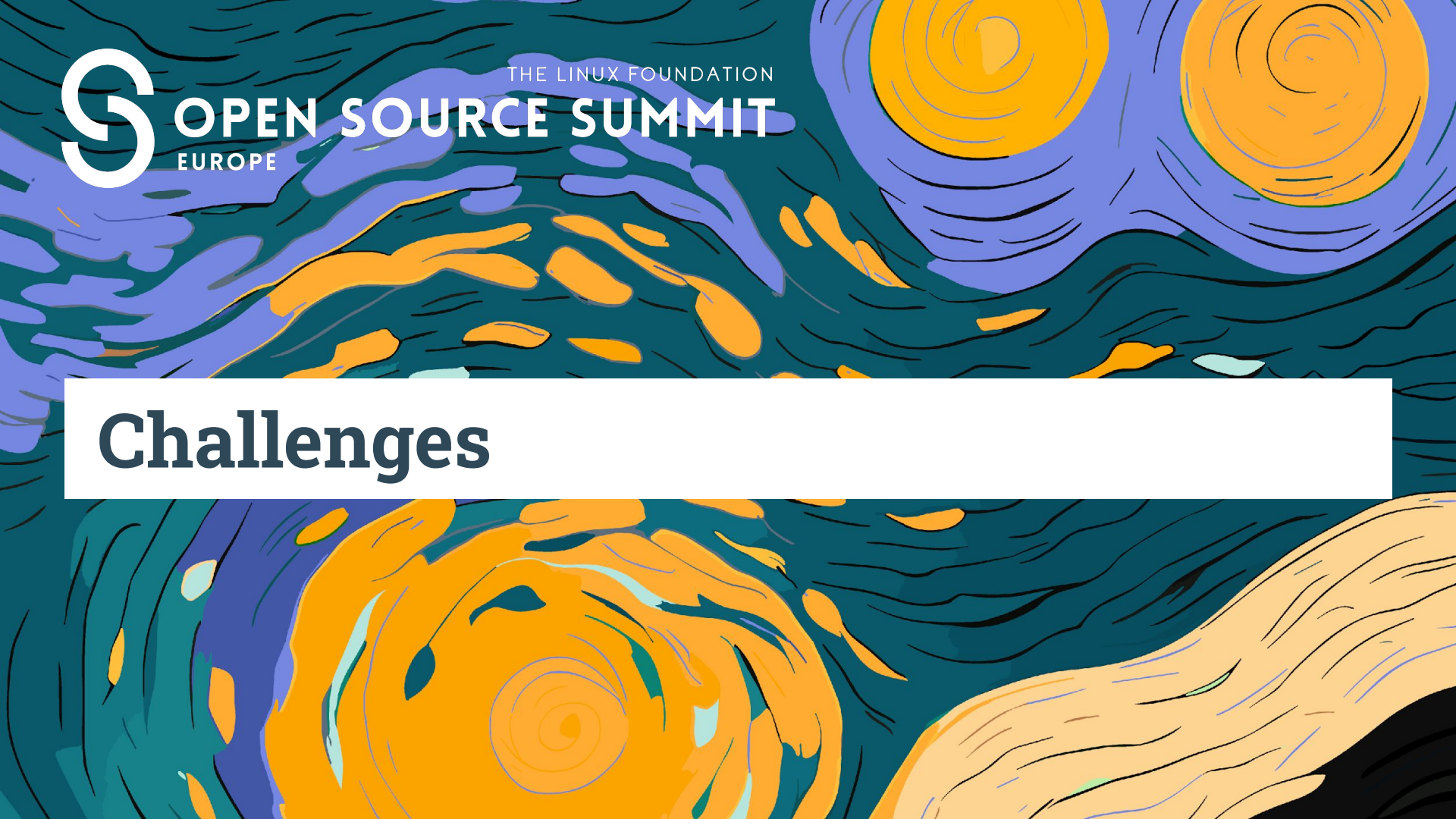
Scaling Up, Part 2: Revisiting UBootDriver

- On slide 19, we defined **tftp** under the **boot_commands** field for UBootDriver - just add more, e.g. **emmc** to boot from there
- There's no reason we can't add more states to the strategy, either

```
class Status(enum.Enum):
    unknown = 0
    off = 1
    uboot = 2
    emmc = 3
    tftp = 4
```

```
boot_commands:
  my_emmc: <stuff goes here for booting
from eMMC>
  my_tftp: <stuff goes here for tftp
boot>
```

```
        elif status == Status.emmc:
            # transition to uboot
            self.transition(Status.uboot)
            self.uboot.boot("my_emmc")
            self.uboot.await_boot()
            self.target.activate(self.shell)
        elif status == Status.tftp:
            # transition to uboot
            self.transition(Status.uboot)
            self._set_server_ip()
            self.uboot.boot("my_tftp")
            self.uboot.await_boot()
            self.target.activate(self.shell)
```



Challenges

Challenges, Part 1: General

- **Documentation hard-to-follow sometimes** - more real-world examples, advanced use-cases would be hugely beneficial
- **Expressive, but still verbose** - could use a tool to help quickly generate templates interactively (similar to labgrid-suggest)
- **Expensive/difficult to get access** to some supported hardware devices
  - May have to stick to COTS (e.g. Tim using PoE switches)
- **Storing and using secrets** (passwords, keys) - e.g. HSM, TPM
- Wrapping commands that need **sudo** (device is production without root login)
- **Proper output format** (json vs simple text report vs html)

Challenges, Part 2: Scaling (10, 20, 50 Boards)?

- Covered this a bit, but still hard, **context-dependent**
- **Resource export/definition dilemma** - one exporter + setup per DUT (Tim, others), or multiple DUTs per exporter (Trevor, others?)
  - Simplest setup is coordinator, one exporter config, one env/client config, and a strategy, but this probably won't work well for large board farms
- **Toggling features:** Use u-boot or not (without changing the strategy or resources), Flash or not (without changing the strategy or resources)
- **How generic can exporter, client configs, and strategies get?**
  - **Making client config generic may be less ideal than being concise**
- **Cop-out answer:** It's Python, just write another abstraction layer

Future Work

Future Work, Part 1

- **Incorporating labgrid workflows in everything we do**
- **Goal:** Publicly available Yocto Project distro with gitlab-ci and gitlab-runners showing the test results with all of the gitlab metadata and labgrid metadata
- **Goal:** Same as above, but with Isar or Buildroot
- **Adding more targets** to our labs (each board has idiosyncrasies about how you hook it up for automation)
  - **Trevor:** Zedboard (Digilent/Xilinx), TBD RISCV64 dev board meeting RVA23 standard
  - **Tim:** Jetson Orin Nano (NVIDIA), BeagleV-Ahead (BeagleBoard)
- **SDLL/CIHIL for some kernel drivers**
  - BeaglePlay + Waveform Click (or other MIKROE): boot built kernel, generate waveform, test values, capture from oscilloscope, iterate

Future Work, Part 2

- Labgrid-planner (Protoyped with Claude)?
- “Gridtop” (htop/k9s for labgrid)?
- Fix the bracketed paste mode issue - some built-in checking
- KernelCI integration?
  - Contributing results to KCIDB
  - Triggering builds/test runs from KernelCI notifications
- Maybe some hardware tests in the Yocto AB (someday)
- More scalability/generic strategies
- **Build + document full examples using only COTS control hardware (Tim’s is one!)**

Special Thanks

- The Labgrid developers
- Kevin Hilman, Jérôme Brunet
- Jan Lübbe

Questions?

Come see us at the booth for a demo!

APPENDIX

References

- https://github.com/labgrid-project/labgrid
- https://github.com/threexc/labgrid-power-up/
- https://gitlab.com/moto-timo/build-beagleplay-kernel/
- https://gitlab.com/moto-timo/beagleplay-isar/
- https://github.com/moto-timo/ansible-role-labgrid
- https://github.com/moto-timo/ansible-role-sd-mux-ctrl
- https://github.com/moto-timo/ansible-role-pdudaemon
- https://github.com/moto-timo/pdudaemon/tree/netgearplus

Abstract (Itemized)

How do you automate your day-to-day embedded software development workflow - if you do? In today's fast-moving world, automation is all-but-essential. In addition to development speed and efficiency, we need it to be confident that our changes are not breaking anything and to continuously keep up with upstream changes and security fixes.

1. Toward this objective, we will explain how to get started with a GitLab runner for continuous integration, and combining it with Labgrid and pytest for automating testing on hardware at your desk.
2. Our target device will be a BeaglePlay booting a Debian OS with the goal of a tight development loop for kernel drivers and devicetrees.
3. We will also discuss how to incorporate that into a more "product" focused testing loop with the help of the Yocto Project.
4. Most importantly, we'll explain why this is such a power-up for the developer by sharing our own experiences and how they've been improved by adopting the workflow we've described.
5. After automating your personal development workflow, we'll explore community initiatives like KernelCI that are using similar tools for automated builds & tests at scale.

Trevor's System Setup (Part 1)

- **Host ("ecogrid") OS:** Debian 12 (Bookworm)
  - **Pros:** Stable, commonly-used packaging ecosystem, sysfs GPIO access
  - **Cons:** Python packages likely to be quite old, need a virtualenv
- **Target OS (eMMC):** Beagleboard.org BeaglePlay Debian 2025-03-25 Minimal Flasher Image
- **Host Python:** 3.11.2 (Debian 12 default)
  - Creating virtualenvs with "python3 -m venv labgrid-venv"
- **(Host) Labgrid:** master branch
  - **Pros:** Latest features
  - **Cons:** Not necessarily as stable (but still pretty good)
- **udev rules for the BL Copilot and gpio access**
- **Tim also uses some Ansible roles for streamlining setup**

Trevor's System Setup (Part 2)

- Set up labgrid with:
  a. sudo apt install -y python3-venv
  b. python3 -m venv labgrid-venv
  c. source labgrid-venv/bin/activate
  d. pip install labgrid
- You should now have access to **labgrid-coordinator**, **labgrid-exporter**, **labgrid-client**, etc.
- **What do we need to start using it?**
  a. Device identifiers from our system's perspective
  b. Functional grouping of devices ("resources") in a structured way
  c. Expression of how to interact with them ("drivers" and "strategies")
  d. One or more scripts (or test suites!) to actually do it

Why:

- **BeaglePlay:**
  - Common, relatively-new, low-cost, variety of interfaces
  - Boots the genericarm64 MACHINE (Yocto Project)
  - Basically upstreamed (mainline kernel-friendly)
- **Labgrid:**
  - Expressive, supports many types of hardware, simple-ish (compared to LAVA)
- **Debian:**
  - Well-understood, stable
  - beagleboard.org images available for eMMC
- **GitLab CI:**
  - More familiar pipeline syntax (to us) versus GitHub Actions, CircleCI, Jenkins, etc.

Software Tools

https://www.beagleboard.org/bb-imager

Using labgrid-suggest

- Monitors the system for new devices and helps you figure out how to match against them
    - Won't work for devices we can't hot plug
- Example: Trevor ran labgrid-suggest, then while it was running he plugged a Raspberry Pi Debug Probe into **ecogrid**, and saw:

```
=== added device ===

  USBSerialPort for /devices/pci0000:00/0000:00:08.1/0000:02:00.3/usb1/1-3/1-3.1/1-3.1.3/1-3.1.3:1.1/tty/ttyACM0
  === device properties ===
  device node: /dev/ttyACM0
  udev tags: , systemd
  vendor: Raspberry_Pi
  model: Debug_Probe__CMSIS-DAP_
  revision: 0101
  === suggested matches ===
  USBSerialPort:
        match:
        ID_PATH: pci-0000:02:00.3-usb-0:3.1.3:1.1
  ---
  USBSerialPort:
        match:
        ID_SERIAL_SHORT: E663AC91D35C5B2E
        ID_USB_INTERFACE_NUM: '01'
  ---
```

Exporter Configuration: the BeaglePlay Resources (Part 1)

- (On the exporter) Matching a USB Serial Converter by ID_PATH

```
$ udevadm info $(tio --list-devices) | grep ID_PATH
E: ID_PATH=platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.4:1.0
```

- Exporter Configuration

```
$ cat /etc/labgrid/exporter.yaml
beagleplay-1:
  location: 'c4labs-chassis-1'
  USBSerialPort:
    match:
      ID_PATH: 'platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.4:1.0'
```

- Verify

```
$ labgrid-client -p beagleplay resources
beagleplay-raspberrypi4-64/beagleplay-1/NetworkSerialPort
```

Ansible

https://www.jeffgeerling.com/blog/2022/ansible-playbook-upgrade-ubuntudebian-servers-and-reboot-if-needed

Mac OS

```
brew install ansible
```

NOTE: if you want sudo no password, the files in /etc/sudoers.d/ are applied in alphabetical order and your file with NOPASSWORD:ALL must be at the end (after `admin` user on beagleplay for instance so preface it with zz_)

Ansible roles

https://github.com/moto-timo/ansible-role-labgrid
https://github.com/moto-timo/ansible-role-pdudaemon
https://github.com/moto-timo/ansible-role-sd-mux-ctrl

Controlling PoE power on a Netgear GS305EPP

wget https://github.com/nitram509/ntgrrc/releases/download/v0.12.1/ntgrrc-v0.12.1-linux-arm64
sudo mv ntgrrc-v0.12.1-linux-arm64 /usr/local/bin/
sudo ln -sf /usr/local/bin/ntgrrc-v0.12.1-linux-arm64 /usr/local/bin/ntgrrc
sudo chmod +x /usr/local/bin/ntgrrc-v0.12.1-linux-arm64
ntgrrc login --address=10.0.0.200
ntgrrc poe status --address=10.0.0.200
ntgrrc poe settings --address=10.0.0.200
ntgrrc poe set -p 3 --power disable --address=10.0.0.200
ntgrrc poe set -p 3 --power enable --address=10.0.0.200

Determining the serial number of your SD-Wire device

Install sd-mux-ctrl and make sure your user has permissions (e.g. on Debian you are a member of sd-mux-ctrl group `sudo adduser $USER sd-mux-ctrl`). If you installed sd-mux-ctrl and have not rebooted yet, do so first or you will see libftdi errors.

```
$ sd-mux-ctrl --list
Number of FTDI devices found: 1
Dev: 0, Manufacturer: SRPOL, Serial: YYYYMMDDNNNN, Description: sd-wire
```

Where YYYYMMDDNNNN is a serial number, like 202405010001n (depending on who programmed the SD-Wire, the format of the serial number may be different)

Exiting labgrid-client console

`Ctrl-\` (to get to the microcom menu)

```
Enter command. Try 'help' for a list of builtin commands
-> quit
exiting
```

Or:

`Ctrl-\` (twice)

Configuring /etc/pdudaemon/pdudaemon.conf

Will be used as "pdu" in env.yaml

```
...
    "netgearplus": {
        "driver": "netgearplus",
        "ip": "10.0.0.1",
        "port_count": 4,
        "password": "s3cr3t"
    }
...
```

ip addr of the netgear PoE capable switch

https://github.com/moto-timo/pdudaemon/tree/netgearplus

https://github.com/moto-timo/py-netgear-plus/tree/debian/latest

Configuring PDUDaemonPort in /etc/labgrid/exporter.yaml

```
...
  PDUDaemonPort:
    host: 'beagleplay-raspberrypi4-64.local'
    pdu: 'netgearplus'
    index: 3
...
```

"Host" that is running pdudaemon.service

Name of the pdu entry in /etc/pdudaemon/pdudaemon.conf

POE Port number

Configuring UBootDriver for Beagleplay minimal Debian 12.11 image

```
UBootDriver:
  prompt: '=> '
  autoboot: 'Press SPACE to abort autoboot in 2 seconds'
  interrupt: ' '
  name: 'bp-uboot'
  boot_commands:
    tftp: 'setenv bootargs "$bootargs console=ttyS2,115200n8
root=/dev/mmcblk1p3 ro rootfstype=ext4 resume=/dev/mmcblk1p2
rootwait net.ifnames=0 quiet"; setenv kernel_comp_addr_r
0x85000000; setenv kernel_comp_size 0x2000000; tftp
0x80000000 beagleplay/Image.gz;tftp 0x82000000
beagleplay/k3-am625-beagleplay.dtb; booti 0x80000000 -
0x82000000'
```

https://bootlin.com/doc/training/embedded-linux-beagleplay/embedded-linux-beagleplay-labs.pdf

Installing KCIDB (in a virtualenv)

```
sudo apt install postgresql
sudo apt install libpq-dev
pip3 install 'git+https://git@github.com/kernelci/kcidb.git'
```

Kevin's Dry-Run #0 Feedback

- ~~Clarify serial port matching doesn't need index if only concerned about one port~~
- Highlight "places" as abstractions of one or more boards, interfaces - a way to not care about implementation specifics (beyond that you have the right board)
- ~~Emphasize that even with our different setups, we ultimately get to use the same set of control commands~~
- ~~change "tftp" and "emmc" targets in UBootDriver boot_commands to "my_tftp" and "my_emmc"~~
- Revisit abstracting variables for boot_commands list (can we hard-code less?)
- Time to Automate (Part 3) - highlight that this is GitLab CI YAML
- Why Do We Need All of This? - highlight that the same minimal tooling is applicable locally and in CI systems
- ~~Skip "Results" slide - encourage visiting at the booth~~
- Challenges (Part 2) - compare to LAVA - no cop-out, mention how labgrid could better abstract providing/setting boot variables
- Note: click board driver would need to go into a ramdisk as a module, not full kernel
- Tim's Story about before and after - bring to the forefront, reference it regularly
- ~~Revisit goal in every section~~
- (If possible) clarify the labgrid terminology a bit - resource vs driver vs strategy, and where everything goes

GPIO Evolution in Kernel - Relevant to Labgrid

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/drivers/gpio/gpiolib-sysfs.c?id=2028f854b3f5b3816cd5d5dd83057a873eddc4d6