Comcast / Sky HAL Testing POC (3PE Progress)

By Gerald Weatherup

V1.1

24/5/22

What is the HAL?

- HAL is a Hardware Abstraction Layer between RDK & Vendor Software
  - It defines the requirements for the Vendor to integrate into the RDK
  - It defines the Service Level Agreement between the RDK and Vendor Software
  - It defines a common interface for the RDK upper layers

- HAL – Implementation
  - Is created by the Vendor (SoC Providers, OEM etc.)
  - It can be developed on the Vendor’s reference platform
  - Will be tested on the Vendor’s reference platform
  - Is expected to run on Sky / Comcast hardware, after bring-up
  - It may also contain common or open source components in order to operate

Key Requirements

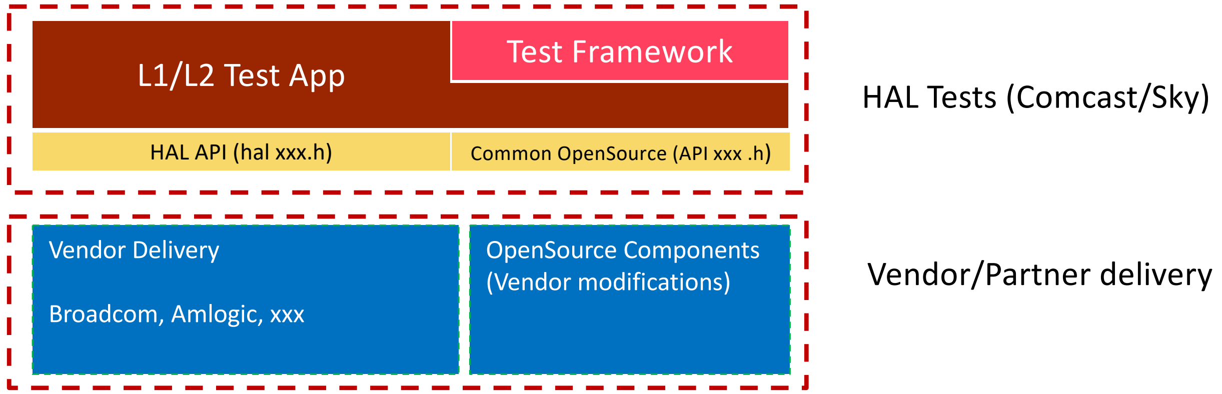
- Test the vendor delivery
  - Provide a robust test and certification suite to the vendor
  - Ensure that no upper RDK layers or RDK infrastructure is required
  - Test the HAL Interface Specification has been met
- Improve the quality of the delivery
  - Remove assumptions about quality, and use metrics to prove quality
  - Remove and reduce regressions
  - Improve testing with feedback loops
- Documentation
  - Documentation that provides requirements for each interface with no ambiguity.
  - Documentation that details system requirements, how is the interface used? state machine expectations? threading requirements?
  - Detailed documentation on the testing suite.
  - Templates are now available, (ongoing progress for RDK-B), via the drive from Peter Stieglitz & the RDK–V Team
- Versioning
  - HAL, Test suit and documentation will be versioned together, each component will have individual versions
    - V3.0 - Wifi-HAL RDK-B currently contains 105 deprecated APIs to support older platforms, this needs to be corrected with delivery versioning.

How are we going to achieve this?

- Improve the quality of the software deliverables via automation
  - Unit / API Function testing (Level 1)
    - Make sure the individual parts work correctly on their own
    - Ensure that the inputs result in the correct output
    - Ensure the foundation is correct
    - Write test cases for positive/negative, max/min, and repetitive calls, for each API interface
  - Component Testing (Level 2)
    - Validate the function of individual components
    - Validate the performance and operation of the component as a whole to meet expectations
      - Start the interface, reset the interface, close the interface, set/get params etc.
      - Performance testing, Load the system how well does the interface perform
      - Feedback from reported issues during testing and infield, write test cases to exercise functionality.
      - Valgrind (Memory testing), Longevity Testing etc.
  - Deliver Quality
    - Use standard opensource frameworks to generate metrics in a consistent manner.
    - Use the metrics to identify issues and areas of focus.
    - Documentation will be included by Doxygen, which will generate HTML & PDF output
    - Functions without tests should not happen
    - Open feedback loops to improve quality
      - Feed data back into tests to improve software robustness
      - Create reproduction scenarios as required to aid issue investigation.

How does it work?

- Delivery contains Test applications & Test Framework, Stubs, HAL API headers, and versions of the open-source components as required.
  - All components are versioned
  - Documentation is generated from the header files
  - The vendor is free to link against his own delivery and OpenSource layers as required.
  - Linux stubs are provided to allow rapid development and create new tests in isolation

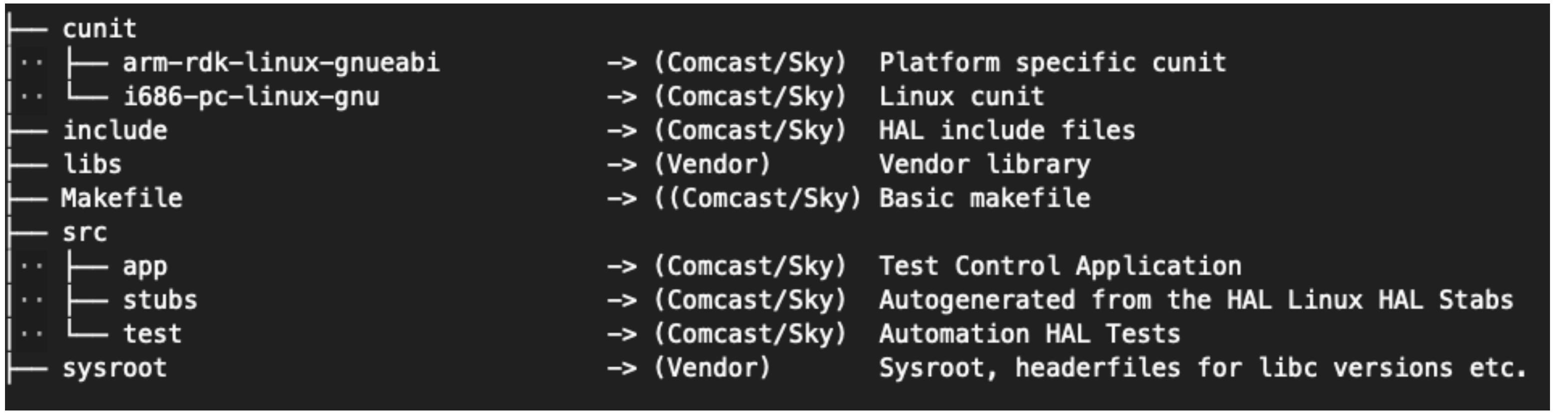


Proof of Concept - Demo

- There are many open frameworks available
  - https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
  - Unity - http://www.throwtheswitch.org/unity
    - Mocking - Describe
    - Works well for mocking, similar to Google Test
    - Provides Proof of coverage via plugins, and multiple output formats.
    - Not so good on the actual platform, required to port to the platform ideally would require ruby on the destination.
  - CUnit was selected for this POC
    - http://cunit.sourceforge.net/index.html
    - Simplicity, painless, run locally
    - Has multiple operation modes, and can run the independent application directly on the target easily.
    - Low-Level engineers who should be writing these tests, are predominantly C engineers
    - Vendor deliveries are generally in native C
  - Other frameworks being considered
    - Ongoing discussions with Comcast teams
    - Google Test, is currently in use in the upper layers of the RDK and is used in C++ to mock layers for testing

How does it work?

- Example CUnit Framework delivery
  - Download CUnit http://cunit.sourceforge.net/screenshots.html
  - CUnit is build with the required toolchain via config change, in the example below build arm & i686.
  - Create test app and tests
  - Example directory layout and ownership
  - Add documentation directory



```
├── cunit
│ ·· ├── arm-rdk-linux-gnueabi           -> (Comcast/Sky)  Platform specific cunit
│ ·· └── i686-pc-linux-gnu               -> (Comcast/Sky)  Linux cunit
├── include                              -> (Comcast/Sky)  HAL include files
├── libs                                 -> (Vendor)       Vendor library
├── Makefile                             -> ((Comcast/Sky) Basic makefile
├── src
│ ·· ├── app                             -> (Comcast/Sky)  Test Control Application
│ ·· ├── stubs                           -> (Comcast/Sky)  Autogenerated from the HAL Linux HAL Stabs
│ ·· └── test                            -> (Comcast/Sky)  Automation HAL Tests
├── sysroot                              -> (Vendor)       Sysroot, headerfiles for libc versions etc.
```

How does it work?

- Wifi – Hal Example of scope (V3 header files)

Header	Control Area	Functions
wifi_hal_ap.h	AP Interfaces	58
wifi_hal_client_mgt.h	Client Manager	31
wifi_hal_deprecated.h	Old Deprecated API's	105
wifi_hal_extender.h	Plume Mesh / Extender API	20
wifi_hal_generic.h	Generic Init/ Reset / Led	7
wifi_hal_radio.h	Configure / Read / reset Radio	34
wifi_hal_sta.h	STA VAP Control Interface	5
wifi_hal_telemetry.h	Wifi measure & telemetry	10
Total		288

Demo

- Framework works under Linux with stubs
  - Shows up issues with the definitions of the HALs
  - Useful for prototyping testing framework, and removing errors
  - Future: Could be upgraded to provide a simulator, to exercise the RDK Layers
- Framework on RDK-B (Ada SR300) Platform
  - Manual or automation testing suite is possible
  - No infrastructure is required
  - Console, Basic, or Automated runs available
- Example documentation issues
  - Doxygen comments does exist and is incomplete as will be shown.
  - No information on the component running execution requirements
    - E.g. Power Management, Memory requirements, Quality control etc.

Versioning

- HAL versioning is required
  - HAL means header files which include documentation, and a testing suite.
  - Lock Step, Documentation, Testing and API versions need to be maintained
  - Fixed release versions/upgrades moving forward
  - Tests should be written based on the documentation & API Upgrade to test the HAL Layers.
- Versioning proposal
  - A standard version model is proposed
    - <Major>.<Minor>.<Documentation>
      - Minor is backwards compatible
      - Major is not backwards compatible
- Ongoing discussions on best practices to isolate RDK upper layers (Pete Stieglitz, etc.)
  - Capabilities upgrade, so upper layers can dynamically check features or capabilities
  - ABI interface to return state unsupported functions

Federated Delivery Model

- Fixed version release model
  - Comcast call this delivery model “federated”
  - Standard industry practice and used by Google, BT, Virgin, Sky etc.
- Allows control of adoption
  - Products choose to take versions of the HAL to use
    - Bugfixes can be applied to the latest or specific versions of the HAL
    - Removal of redundant APIs can be part of the roadmap
  - Vendors implement specific versions of the HAL
  - Documentation is available for Specific Versions of the HAL
  - HAL moves forward as required based on future requirements, without affecting any project.
  - Ensures Build Stability
  - Legacy platform management needs to be considered

Next Steps

- Documentation improvements
  - Templates and perform a systematic review of all HALs being driven by Pete Stieglitz, and the RDK-V Team
  - Needs to be adopted by the RDK-B team, and driven.
- Review the TDK implementation
  - TDK has stated that they're currently testing the HAL, a review of this will take place this week.
    - There is a good place for TDK testing, but there are problems currently.
      - Depending on RDK infrastructure
      - Unstable, build system breaking
      - TDK is not directly testing the vendor delivery
      - Driven by changes behind curve on development teams.
      - Tests should be written with the HAL changes, to test the HAL changes, then released for general consumption
- Review Google Test Framework
  - Work with Dominic D'Souza on the Google test framework use cases
    - Initial thoughts are mostly used to mock and isolate implementation laying, rather than testing the requirements on the box
    - Will this be adaptable for the HAL Testing use case, further investigation is required.

Next Steps

- External Stimulus Testing (Out of the box, Level 3 Testing)
  - Whitebox testing (Akin to Sky Q (QT Test Applications))
    - Pushing the platform performance without the RDK to ensure a stable platform
    - Load the system past its limits, fail faster
    - Worst case scenario testing, video, audio, content security, gstreamer, multi-decode
    - Platform profile should be defined
    - Longevity, memory budgeting, performance
    - DDR Bandwidth Testing
    - System profiling
    - Useful for performance on external influences
    - Multiple devices can be involved