Tock Hardware CI

Project Specification

Jefferson Chien, Arvin Lin, Jack Sheridan

Project Charter

Project Overview

Feature updates and bug fixes happen frequently in the software development cycle, and if a thorough test could be performed every time a developer creates a feature or resolves a bug, the developers will know whether they can publish the new changes, or they need to edit the software and resolve the issues. However, Tock OS currently does not have a workflow setup to test its validation on target hardware. So this project's goal is to create a solution that fits the workflow described above and can perform tests on embedded devices.

Project Approach

The goal of the project is to have the developer see the test result soon after they have updated or changed the Tock repository on GitHub. Since the goal also emphasizes the test to be run on the embedded device, a medium to integrate the GitHub repository and the embedded devices is required, and this project will use a Raspberry Pi for that purpose.

In order for a Raspberry Pi to receive updates from Tock repository and to build and install the Tock OS image onto an embedded device, GitHub Action Server, which will be used to track the Tock repository for pull requests and updates, will be used to initialize the workflow. J-Link will be used to establish a connection between Raspberry Pi and an embedded device. In addition, to simplify the project to a manageable scale, this project will be using the nrf52840 development kit. The Raspberry Pi will also be used to monitor the activity of the embedded device - nrf52840dk - and it will be connected to the board with jumper wires and use the GPIO Zero library in python to track the value of each GPIO pin. This will allow the project to be automated with scripts, and that brings us to the streamline of the workflow.

The majority of the work will be scripted through bash, such as starting compilation and executing installation, and the remaining parts that contain creating reports and formatting the log output shall be done in python. After the python script has created the log, it can simply print the log into the file descriptor 1, which would be piped back to the GitHub Action output.

Minimum Viable Product

Our minimum viable product is a complete closed-loop hardware CI. This can be defined and tested by showing that, after a Git push or Github accepted pull request, the Raspberry Pi is triggered to pull the newest code and compile it. After compilation, the Raspberry Pi then flashes this new Tock image onto some piece of hardware. (In our MVP case, a nRF52840). The Raspberry Pi then flashes a test application onto the hardware, and the hardware will then run that test app sequentially. The Pi will listen to it's pins for the correct expected result from the hardware, and report back to the Github CI system to close the loop. The idea behind the MVP is not for large or complicated tests, but instead a small simple test that proves the test harness actually works.

The MVP will consist of the hardware harness: an internet connected Raspberry Pi running a Github action runner listening for requests from Github. The Pi will also have all executable code for compiling, flashing, running, and listening to the tests. (This will consist of bash shell scripts and python scripts among others). The Pi will be connected to a nRF52840, both via the usb port for direct connection to the device, as well as via GPIO pins for the running of the tests.

After we complete the MVP we will first iterate on the complexity and usefulness of the CI output. At first, we would like to simply "dump" the terminal output into Github, but later, we would like to add incremental descriptive "pass/fail" tests that the Github action runner can report on. After that, we can iterate further by expanding on our tests and we can test more rigorously and begin testing actual features of the operating system.

Finally, the longer term goals and bigger image is to be able to have a PI running for each and every board that Tock supports. These test harnesses should also be able to have some remotely configurable list of tests that can be changed and appended to based on the needs of the hardware. Eventually, due to the inexpensiveness of the Raspberry Pi, the idea is to bundle the entire test system into a ready-to-go operating system image for the Pi, so possible Tock users can set up their own Pi with potentially proprietary hardware and allow Tock developers to see how the system is running.

Risk Assessment

The biggest risks in this project happen to be in the interface between the Pi and the hardware itself. Github implementation and the actual compilation of Tock shouldn't be too difficult, because it should work with any computer regardless of if it is a Pi or not. The problem arises when we have to flash a (potentially faulty) OS onto a piece of hardware, run an application, and listen to pins. Many things could happen including: failure to boot the OS, failure to install the application, failure or "freezing" during

application runtime, as well as many others. These edge cases all have to be dealt with cleanly and automatically. There is not always going to be someone sitting at the test harness because it is automated, so it is important that all problems are solved right away and do not block Tock development. These risks and problems can be solved by possibly cutting power to the machine, retrying the process among others, but it is important to realize the possibility of these being a blockade in feature addition, and therefore should be ironed out immediately after construction of the MVP.

Group Management

Within our group, Arvin provides most of the initial proposals regarding project directions and plans. However, each of the group members contributes to group management and comes to an agreement through discussions. We have a Discord channel setup for discussions and communication. Furthermore, for group members, we will hop on the Discord channel and work synchronously every Saturday and the actual time will be determined by each member's schedule. We would also meet Professor Pannuto every other week on Monday to report our progress and receive feedback regarding our work and further plans. In addition to the discord channel, we could consult and contact Professor Pannuto via the Slack channel for Tock OS.

To make sure our project goes smoothly and stays on schedule, we have determined several milestones. This would be further discussed in the *Project Milestones and Schedule* section. For each milestone, all three group members should have their assigned task completed. If someone fell behind, the synchronous work meeting on Saturday would be a good chance to ask for help and receive support. Last but not least, if the schedule really slips, we would adjust the schedule and distribute tasks to make sure the next milestone can be delivered on time with good quality work.

Project Development

Development Roles:

- Arvin: Hardware specialist
- Jack and Jeff: Github/Tock integration

Hardware:

- Nordic nRF52840 development kit (Sent to each of us by Pat Pannuto!)
- Raspberry Pis (Each of us already had one!)
- Jumper Cables (Ordered and on their way)

- USB power control modules (Low priority)

Software:

- Tock OS (Open source!)
- Tock Loader (Open source!)
- Github actions runner (Open source!)
- Ubuntu Server 20.04 (Open source!)

Testing:

- Manual testing of the hardware vs automated via Github
- Manually update a “fake” clone of the Tock Operating system to simulate a true Github triggered event.

Documentation:

- All done inside of the tock-test-harness markdown files.

Project Milestones and Schedule

Milestones: (Low level)

1. Installation of Tock OS on nRF52840 dev kit
2. Compilation and installation of sample Tock OS testing App on nRF52840 dev kit
3. Ubuntu server setup on Raspberry Pi
4. Test Github runner on Raspberry Pi and nRF52840 dev kit
5. Setup GPIO testing software
6. Setup mechanism to run testing Apps on the embedded device
    a. Generate configuration files for test specification
    b. Create and design workflow to integrate config files into custom tests
7. Record behaviors to generate logs which is then send back to Raspberry Pi
8. Compile all logs, including compilation results and messages on GPIO, into a single compressive report on Raspberry Pi.
9. Automatically upload CI results onto the Github repository under the CI event in a reasonable format.
10. Construct new tests or adapt more existing Tock OS test
11. (Optional) Make logs look appealing. Also provide real time reports.
12. (Optional) Setup embedded system boot controller

Milestones: (High level deliverables)

Date Done	Deliverable	Specifications
Apr 19	Development environment. Autonomous update and compilation	Arvin: - Milestone 1 & 2

	mechanism.	Jack & Jefferson: - Milestone 3 & 4
May 03	First MVP	Jack: - Milestone 9 Arvin & Jefferson: - Milestone 5, 6, 7 & 8
May 17	Extended MVP	Jefferson: - Milestone 10, 11 Arvin & Jack: - Milestone 10, 12

Schedule Calendar

Mon	Tue	Wed	Thu	Fri	Sat	Sun
Apr 19 Deliverable 1	Apr 20	Apr 21	Apr 22	Apr 23	Apr 24 Jack: M9 Arvin & Jefferson: M5, M6	Apr 25
Apr 26	Apr 27	Apr 28	Apr 29	Apr 30	May 1 Jack: M10 Arvin & Jefferson: M7, M8	May 2
May 3 Deliverable 2	May 4	May 5	May 6	May 7	May 8 Jefferson: M10 Arvin & Jack: M10	May 9
May 10	May 11	May 12	May 13	May 14	May 15 Jefferson: M11 Arvin & Jack: M12	May 16
May 17 Deliverable 3	May 18	May 19	May 20	May 21	May 22	May 23